

SANDIA REPORT

SAND2000-8877

Unlimited Release

Printed October 2000

Some Parallel Extensions to Optimization Methods in OPT++

V. E. Howle, S. M. Shontz, and P. D. Hough

Prepared by
Sandia National Laboratories
Albuquerque, New Mexico 87185 and Livermore, California 94550

Sandia is a multiprogram laboratory operated by Sandia Corporation, a Lockheed Martin Company, for the United States Department of Energy under Contract DE-AC04-94AL85000.

Approved for public release; further dissemination unlimited.



Sandia National Laboratories

Issued by Sandia National Laboratories, operated for the United States Department of Energy by Sandia Corporation.

NOTICE: This report was prepared as an account of work sponsored by an agency of the United States Government. Neither the United States Government, nor any agency thereof, nor any of their employees, nor any of their contractors, subcontractors, or their employees, make any warranty, express or implied, or assume any legal liability or responsibility for the accuracy, completeness, or usefulness of any information, apparatus, product, or process disclosed, or represent that its use would not infringe privately owned rights. Reference herein to any specific commercial product, process, or service by trade name, trademark, manufacturer, or otherwise, does not necessarily constitute or imply its endorsement, recommendation, or favoring by the United States Government, any agency thereof, or any of their contractors or subcontractors. The views and opinions expressed herein do not necessarily state or reflect those of the United States Government, any agency thereof, or any of their contractors.

Printed in the United States of America. This report has been reproduced directly from the best available copy.

Available to DOE and DOE contractors from

U.S. Department of Energy
Office of Scientific and Technical Information
P.O. Box 62
Oak Ridge, TN 37831

Telephone: (865)576-8401

Facsimile: (865)576-5728

E-Mail: reports@adonis.osti.gov

Online ordering: <http://www.doe.gov/bridge>

Available to the public from

U.S. Department of Commerce
National Technical Information Service
5285 Port Royal Rd
Springfield, VA 22161

Telephone: (800)553-6847

Facsimile: (703)605-6900

E-Mail: orders@ntis.fedworld.gov

Online order: <http://www.ntis.gov/ordering.htm>



Some Parallel Extensions to Optimization Methods in OPT++

V. E. Howle¹, S. M. Shontz², P. D. Hough³

Computational Sciences and Mathematics Research Department, MS 9217
Sandia National Laboratories
Livermore, CA 94551-0969

ABSTRACT

OPT++ provides an array of optimization tools for solving scientific and engineering design problems. While these tools are useful, all of the code is serial. With increasingly easy access to multiprocessor machines and clusters of workstations, this results in unnecessarily long times to solution. In order to correct this problem, we have implemented a number of parallel techniques in OPT++. In particular, we have incorporated a speculative gradient algorithm that drastically reduces the time to solution for standard trust-region and line search algorithms. In addition, we have implemented a new version of the Trust-Region Parallel Direct Search (TRPDS) algorithm of Hough and Meza that yields a significant reduction in solution time for problems with expensive function evaluations.

Keywords: parallel optimization, nonlinear programming, optimization software

¹Email: vehowle@ca.sandia.gov. Part of this work performed while the author was a member of The Center for Applied Mathematics at Cornell University, supported by NSF grant CCR-9619489, NSF grant DMS-9805602, and ONR grant N00014-97-1-0681.

²Currently located at The Center for Applied Mathematics, 657 Frank H. T. Rhodes Hall, Cornell University, Ithaca, NY 14853. Email: shontz@cam.cornell.edu. This work sponsored in part by the National Physical Science Consortium.

³Email: pdhough@ca.sandia.gov

This page intentionally left blank.

1 Introduction

Optimization of functions derived from the modeling and simulation of some physical process constitutes an important class of problems in many engineering and scientific applications. Often, the computer simulation entails the solution of a system of nonlinear partial differential equations (PDE) in two or three dimensions. Other applications include particle dynamics simulations or problems in chemical kinetics. The main characteristic of these types of problems is that the function evaluation is computationally expensive and dominates the total cost of the optimization problem. Depending on the nature of the application and the solution method employed, there can also be noise associated with the evaluation of the objective function. This noise can usually be reduced, but only at the cost of making the computation time even greater. In many of these applications, derivative information is also not available or must be computed using finite differences, thereby increasing the cost of the optimization and generating noisy gradients. Fortunately, the dimension of the optimization problem in many of these optimal design problems is small (usually on the order of tens of parameters). In this study, we will concentrate on techniques for parallelizing unconstrained optimization algorithms when the number of available processors is comparable to the number of optimization parameters. The rationale for this decision is that although massively parallel computers are available, the majority of computational power in most industrial or scientific settings consists of small clusters of shared memory processors (SMP's) or networks of workstations (NOW's) that can be used in a similar capacity.

Schnabel [10] gave an excellent review of the challenges and limitations in parallel optimization. In that review, Schnabel identified three major levels for introducing parallelism: 1) parallelize the function, gradient, and constraint evaluations, 2) parallelize the linear algebra, and 3) parallelize the optimization algorithm at a high level. There have been many attempts to parallelize nonlinear optimization methods at all of these levels. A description of these efforts can be found in, e.g., [7]. In this study, we choose to focus on the third option due to the characteristics of the problems mentioned above. In particular, the first option is not usually available to us because for many situations we do not have access to the source code for the function or the constraints. In addition, the dimension of the optimization problems of interest is usually small, and therefore parallelizing the linear algebra would not yield any benefits.

In this paper, we consider two classes of parallel optimization methods. The first is a speculative gradient approach introduced by Byrd, Schnabel, and Shultz [2]. This is a line-search algorithm that speculatively computes components of the finite difference gradient at the trial point while the function is being evaluated at that point. Since the trial point is accepted the majority of the time, this can result in substantial computational savings over the non-speculative version of the algorithm. The other class of methods is the Trust-Region Parallel Direct Search (TRPDS) algorithm developed by Hough and Meza [7]. This method combines a trust-region method with a parallel direct search (PDS) method in a way that retains the best properties of both. TRPDS provides computational savings over both the standard trust-region and the PDS methods. In addition, it provides a great deal of flexibility to allow the incorporation of techniques that will result in further savings.

The remainder of the paper is organized as follows. In section 2, we describe the speculative gradient approach mentioned above. In section 3, we describe the TRPDS algorithm and how it fits into the generalized approximation model framework. In addition, we describe a modification to the TRPDS algorithm that promises computational savings over the original implementation. Section 4 contains numerical comparisons of the algorithm on a set of standard test problems, as well as on an engineering application. Section 5, the final section of the paper, contains a discussion of the conclusions and future research directions.

2 Speculative Gradients

One approach to the type of optimization problems considered in this paper is to use a traditional Newton method. This type of method comes in two flavors: line search and trust region. These methods have the benefit of desirable convergence properties; however, they offer few options for parallelizing at a high level. One notable exception was introduced by Byrd, Schnabel, and Shultz in 1988 [2]. Because analytic gradients are not typically available in our applications, it is perfectly reasonable to assume that they will be approximated by finite differences. Byrd et al. suggest a straightforward way to take advantage of multiple processors when using a line search method with finite difference gradients. In particular, extra processors are used to compute components of the finite difference gradient at the trial point while

the function is being evaluated at that point. This is referred to as a speculative gradient approach, and the idea applies equally well in the trust-region setting. We focus on using the speculative gradient idea with a trust-region method for the remainder of this paper.

Recall that in each iteration of a trust-region method, a quadratic model of the objective function, f , is formed, and a region in which the model is trusted to approximate the actual function accurately is determined. A trial step is then computed by approximately solving the following subproblem:

$$\begin{aligned} \min_{\mathbf{s} \in \mathbb{R}^n} \quad & \psi(\mathbf{s}) = g(\mathbf{x}_c)^T \mathbf{s} + \frac{1}{2} \mathbf{s}^T H_c \mathbf{s}, \\ \text{s. t.} \quad & \|\mathbf{s}\|_2 \leq \delta_c, \end{aligned} \tag{1}$$

where \mathbf{x}_c is the current point, \mathbf{s} is the step, $g(\mathbf{x}_c)$ is the gradient of f at the current point, $H_c \approx \nabla^2 f(\mathbf{x}_c)$ is the Hessian approximation at the current point, and δ_c is the size of the trust region. Once a step has been computed, the function is evaluated at the trial point, $\mathbf{x}_t = \mathbf{x}_c + \mathbf{s}$. If there is sufficient decrease in the function value, the gradient and the Hessian are then evaluated at that point.

In a parallel setting with p processors, $p - 1$ of the processors are sitting idle while the function is evaluated at the trial point. The goal of the speculative gradient approach is to remedy this inefficient use of processors in the following manner. While we are evaluating the function at the trial point, use the remaining $p - 1$ processors to calculate up to $p - 1$ components of the finite difference gradient. If the trial point is accepted, we already have $p - 1$ components of the gradient available and only need to calculate the remaining $n - (p - 1)$ components, where n is the dimension of the problem. If the trial point is not accepted, we simply try again at the next trial point. In that case, we have not lost any more computational time than was already required for the function evaluation.

With the speculative gradients change, the trust-region algorithm is as follows.

Algorithm 1. Trust-Region method with Speculative Gradients
Given p processors, \mathbf{x}_0 , \mathbf{g}_0 , H_0 , δ_0 , and $\eta \in (0, 1)$
for $k = 0, 1, \dots$ until convergence **do**
 for $i = 0, 1, \dots$ until step accepted **do**
 1. Find \mathbf{s}_i that approximately solves (1)
 2. Processor 1: evaluate $f(\mathbf{x}_k + \mathbf{s}_i)$
 Processor j : evaluate $g_{j-1}(\mathbf{x}_k + \mathbf{s}_i)$ for $j = 2, \dots, p$
 3. Compute $\rho = (f(\mathbf{x}_k + \mathbf{s}_i) - f(\mathbf{x}_k))/\psi(\mathbf{s}_i)$
 if $\rho > \eta$ **then**
 4. Accept step, set $\mathbf{x}_{k+1} = \mathbf{x}_k + \mathbf{s}_i$
 5. Evaluate $g_j(\mathbf{x}_k + \mathbf{s}_i)$ for $j = p, \dots, n - (p - 1)$
 6. Update H_k
 else
 7. Reject step
 end if
 8. Update δ_k
 end for
end for

Our current implementation only takes advantage of $n+1$ processors, even if more are available. One way to take advantage of additional processors would be to extend the speculative idea to computing as much of the finite difference Hessian as possible. We leave this to future work and for the purposes of this paper, we use a BFGS approximation to the Hessian.

3 TRPDS with Generalized Approximation Models

In order to take advantage of the strengths of both the trust-region (TR) and parallel direct search (PDS) classes of algorithms, Hough and Meza developed the Trust-Region PDS (TRPDS) class of algorithms [7]. TRPDS employs the standard trust-region framework, but uses PDS to solve a non-standard subproblem to compute the step at each iteration. This subproblem is known as the PDS subproblem and is defined as

$$\begin{aligned} \min_{\mathbf{s} \in \mathbb{R}^n} \quad & f(\mathbf{x}_c + \mathbf{s}) \\ \text{s. t.} \quad & \|\mathbf{s}\|_2 \leq 2\delta_c, \end{aligned} \tag{2}$$

where \mathbf{x}_c is the current point, \mathbf{s} is the step, and δ_c is the size of the trust region. It is important to note that the only difference from the standard trust-region method occurs in the computation of the step, \mathbf{s} . In particular, the actual objective function, rather than a quadratic model of that function, is approximately minimized (using PDS) over the trust region.

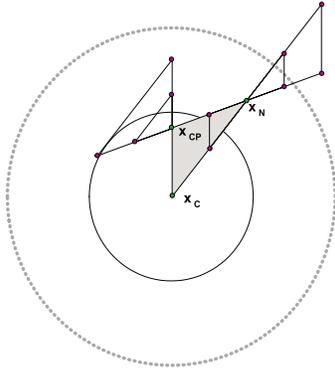


Figure 1: *Overview of the TRPDS algorithm. The point \mathbf{x}_{CP} is the Cauchy point, \mathbf{x}_N is the Newton point, and \mathbf{x}_C is the current point. These points are used to initialize the simplex over which PDS approximately minimizes the function. The solid circle represents the trust region. The step length is allowed to be twice the size of the trust region (dotted circle) to allow for the possibility of taking a step longer than the Newton step.*

Figure 1 illustrates the basic idea of TRPDS. See the paper by Hough and Meza for details of the algorithm [7].

Hough and Meza observed that the TRPDS class of algorithms fits into the generalized trust-region framework of Alexandrov, Dennis, Lewis, and Torczon [1]. The generalized framework provides a great deal of flexibility both in the choice of trust-region model and in the method of computing the step at each iteration. In particular, let a be an approximation to the objective function, f . If

1. $a(\mathbf{x}_c) = f(\mathbf{x}_c)$,
2. $\nabla a(\mathbf{x}_c) = \nabla f(\mathbf{x}_c)$,

and the sequence of steps generated during the optimization satisfies a fraction of Cauchy decrease condition (FCD) according to a , then the standard

trust-region convergence theory implies that this class of methods will converge [1].

The TRPDS algorithm takes advantage of the flexibility in the step computation in the sense that it solves a non-standard subproblem within the trust-region framework. (Though a can be any model that satisfies conditions 1 and 2, TRPDS currently uses the standard quadratic model.) We build on that flexibility by extending the step computation to a two-phase approach that incorporates the use of a second approximation model. The first phase consists of using PDS to find the p best approximate solutions to the following problem:

$$\begin{aligned} \min_{\mathbf{s} \in \mathbb{R}^n} \quad & m(\mathbf{x}_c + \mathbf{s}) \\ \text{s. t.} \quad & \|\mathbf{s}\|_2 \leq 2\delta_c, \end{aligned} \tag{3}$$

where p is the number of processors and m is a computationally inexpensive approximation to the objective function. Notice that this resembles the PDS subproblem except the objective function has been replaced by an approximation model. In the second phase of the step computation, each processor evaluates the objective function at one of these p trial points. The point that yields the lowest function value is returned as the trial point to the trust-region framework and is processed according to the standard trust-region algorithm. We refer to this variation of the TRPDS algorithm as TRPDS(p).

A formal statement of the TRPDS(p) algorithm follows.

Algorithm 1. TRPDS(p)Given p processors, \mathbf{x}_0 , \mathbf{g}_0 , H_0 , δ_0 , and $\eta \in (0, 1)$ **for** $k = 0, 1, \dots$ until convergence **do**1. Solve $H_k \mathbf{s}_N = -\mathbf{g}_k$ **for** $i = 0, 1, \dots$ until step accepted **do**2. Form an initial simplex using \mathbf{s}_N 3. Compute the p best approximate solutions $\mathbf{s}_1, \dots, \mathbf{s}_p$ to (3) using PDS4. Determine $\mathbf{s} \in \{\mathbf{s}_1, \dots, \mathbf{s}_p\}$ that minimizes $f(\mathbf{x}_k + \mathbf{s})$ 5. Compute $\rho = (f(\mathbf{x}_k + \mathbf{s}_i) - f(\mathbf{x}_k))/\psi(\mathbf{s}_i)$ **if** $\rho > \eta$ **then**6. Accept step and set $\mathbf{x}_{k+1} = \mathbf{x}_k + \mathbf{s}_i$, evaluate \mathbf{g}_{k+1} , \mathbf{H}_{k+1} **else**

7. Reject step

end if8. Update δ **end for****end for**

There are many issues to consider in the TRPDS(p) algorithm such as forming the initial simplex and computing the PDS step; however, these issues are discussed in [7] so we do not repeat the discussions here. Instead, we focus on the computational impact of using an approximation model. The results of our numerical study appear in the following section.

4 Numerical Results

In order to evaluate the performance of the extensions to the trust-region and TRPDS algorithms, we chose a standard set of test problems from the literature and an engineering application problem based on a computer model of a chemical vapor deposition furnace.

In all of the tests, gradient approximations were computed using parallel forward differences. Also, the TRPDS(p) algorithm requires an approximation model for the step computation. We used a quadratic model constructed by forming a Taylor series expansion of the objective function, i.e.,

$$f(\mathbf{x}_c + \mathbf{s}) \approx m(\mathbf{x}_c + \mathbf{s}) = f(\mathbf{x}_c) + \mathbf{g}_c^T \mathbf{s} + \frac{1}{2} \mathbf{s}^T H_c \mathbf{s}.$$

Here $\mathbf{g}_c = g(\mathbf{x}_c)$ is the gradient of f at the current point, and $H_c = H(\mathbf{x}_c)$ is the current Hessian approximation.

4.1 Standard Test Problems

We used a number of test problems obtained from papers by Moré, Garbow, and Hillstom [9], Byrd, Schnabel, and Shultz [2], and Conn, Gould, and Toint [3]. For comparison purposes, we solved these problems with the

Table 1: *Test problems.*

| Number | Problem |
|--------|----------------|
| 1 | almost |
| 2 | broyden1a |
| 3 | broyden1b |
| 4 | broyden2a |
| 5 | broyden2b |
| 6 | bv |
| 7 | chain_singular |
| 8 | chain_wood |
| 9 | chebyquad |
| 10 | cragg_levy |
| 11 | epowell |
| 12 | erosen |
| 13 | gen_brown |
| 14 | gen_wood |
| 15 | ie |
| 16 | lin |
| 17 | lin0 |
| 18 | penalty1 |
| 19 | penalty2 |
| 20 | toint_trig |
| 21 | tointbroy |
| 22 | trig |
| 23 | vardim |

standard TRPDS algorithm, TRPDS(p), a standard BFGS trust-region algorithm, and a BFGS trust-region algorithm with speculative gradients. The test problems are listed in Table 1.

The starting points used for the problems were the same as those given in the references. All algorithmic parameters are listed below:

Table 2: *Algorithmic parameters.*

| Parameter | Value |
|----------------------|---------------------------|
| Machine Epsilon | 2.22045×10^{-16} |
| Maximum Step | 1000 |
| Minimum Step | 1.49012×10^{-8} |
| Maximum Iter | 500 |
| Maximum Fcn Eval | 10000 |
| Step Tolerance | 1.49012×10^{-8} |
| Function Tolerance | 1.49012×10^{-8} |
| Gradient Tolerance | 6.05545×10^{-6} |
| LineSearch Tolerance | 0.0001 |

The tests were run on a 64-processor SGI Origin 2000 with the IRIX 6.5 operating system. The step tolerance, the function tolerance, and the gradient tolerance were used as stopping criteria for the optimization algorithms in the spirit of Gill, Murray, and Wright [4].

The results of the experiments for the standard problems appear in Figures 2 through 7. In our applications, the most important measure of performance is the total time to solution of the problem; however, these test problems are extremely inexpensive, making time measurements impractical. Instead, we compare the total number of *concurrent function evaluations* required for each method, where a concurrent function evaluation is defined as in [7]. We feel that this is an accurate reflection of how much time the algorithms require relative to each other since the computational cost of the function dominates the cost of the algorithms in our applications.

First, we compare a standard BFGS trust-region algorithm to the same algorithm with speculative gradient evaluations incorporated. For the speculative gradient method with forward differences, the ideal number of processors is $n + 1$, where n is the dimension of the problem. In this case, we can

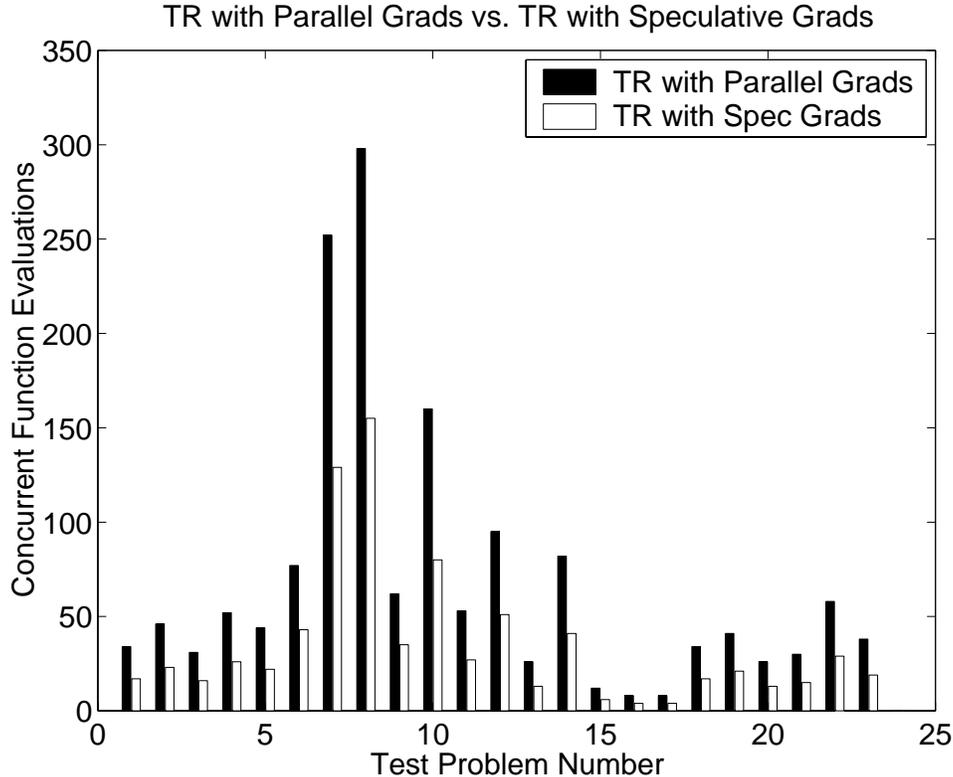


Figure 2: Comparison of BFGS trust-region algorithm and BFGS trust-region algorithm with speculative gradients. The dimension of the problem is 8, and the number of processors used is 9. The BFGS trust-region algorithm with speculative gradients beat the BFGS trust-region algorithm with parallel finite difference gradients.

speculatively calculate the entire gradient each time we evaluate the function at a trial point. So when the trial point is accepted, a second concurrent function evaluation is not required as it is in the non-speculative approach. Note that this implies that we can reduce the number of concurrent function evaluations by up to 50 percent over the standard trust-region algorithm with parallel finite difference gradients. Therefore, for our comparisons of these two variants of the trust-region algorithm, we use $n + 1$ processors.

Results for $n = 8$ are shown in Figure 2. As expected, for all of the test problems, the BFGS trust-region algorithm with speculative gradients needs fewer concurrent function evaluations than the standard BFGS trust-region

algorithm using only parallel finite differences. In the best cases, the speculative gradient variant takes 50 percent fewer concurrent function evaluations. These are the test problems for which we accepted the initial trial point at every iteration. Even in the worst case (problem 9), where the initial trial point was accepted only 76 percent of the time, the speculative gradient approach required 43 percent fewer concurrent evaluations than the non-speculative variant. Schnabel states that in his experience, Newton methods accept the initial trial point about 60 – 70 percent of the time [10]. Based on that observation, as well as on the numerical results presented here, we can expect always to see a substantial benefit to using the speculative gradient variant of Newton methods when finite difference gradients are required.

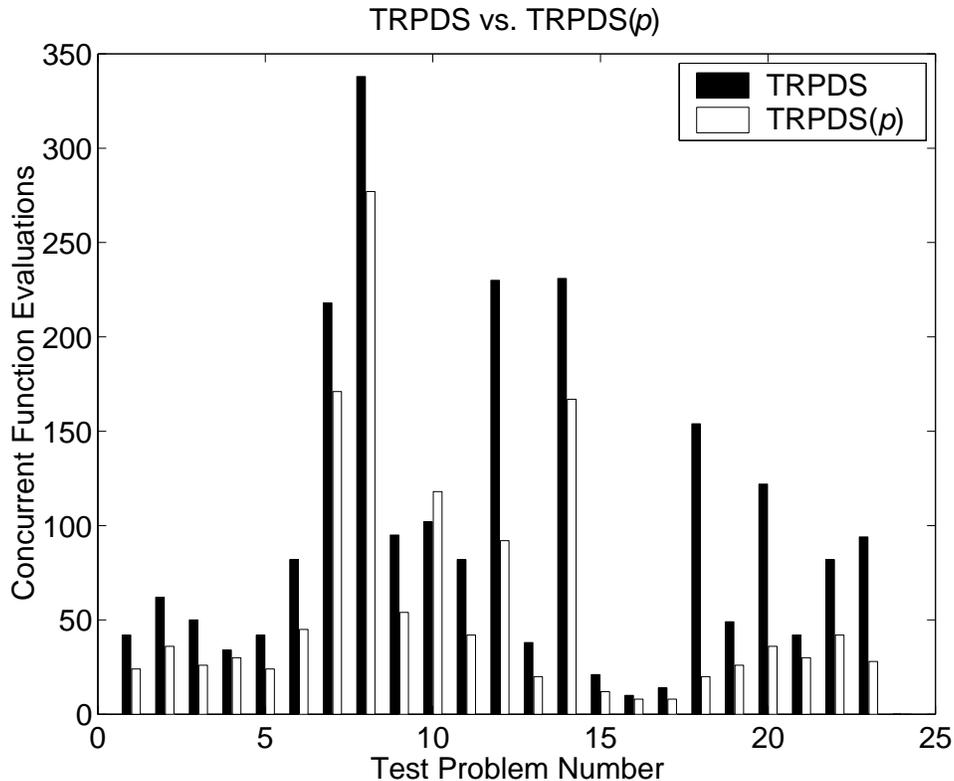


Figure 3: Comparison of TRPDS algorithm and TRPDS(p) algorithm. The dimension of the problem is 8, and the number of processors used is 9. The TRPDS(p) algorithm beat the TRPDS algorithm nearly every time.

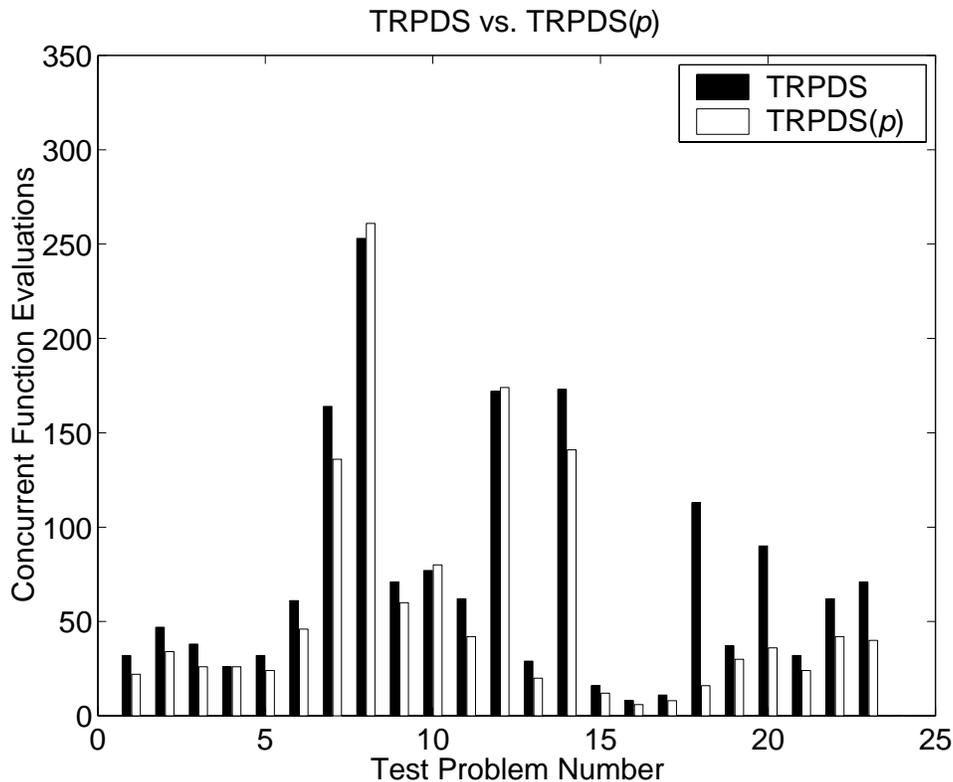


Figure 4: *Comparison of TRPDS and TRPDS(p). The problem dimension is 8, and the number of processors used is 16. TRPDS(p) beat TRPDS on most test problems.*

We next compare TRPDS(p) to the standard TRPDS algorithm. For the standard TRPDS algorithm, the ideal situation is to have the same number of search scheme points (i.e., points at which PDS evaluates the function) as processors so that there is only one concurrent function evaluation per PDS iteration. In the case of TRPDS(p), it is not obvious what the ideal situation is since, by design, there is only one concurrent function evaluation per PDS iteration regardless of the number of search scheme points or processors. Furthermore, it is not clear how the use of an approximation model will affect the performance of the algorithm. For our initial tests, we fixed the search scheme size at $2 \times n$, where n is the dimension of the problem, and we ran tests on $n + 1$ processors and $2 \times n$ processors.

Results for $n = 8$ are shown in Figure 3 (with $p = 9$) and Figure 4 (with

$p = 16$). We note that the concurrent function evaluations reported here are evaluations of the actual (expensive) function. On average, TRPDS(p) takes 36 percent fewer concurrent function evaluations than the standard TRPDS algorithm. In essence, TRPDS(p) replaces some of the expensive concurrent function evaluations done in TRPDS by inexpensive model evaluations. Thus, this new variant is able to reach the solution while incurring a lower computational cost. There are some cases when the original TRPDS does beat TRPDS(p) (e.g., problem 10). In these cases, the quadratic model used by PDS is a particularly poor approximation to the function early in the optimization. For this reason, more iterations, and thus more concurrent function evaluations, are required to reach the solution.

Figures 3 and 4 indicate that the number of processors affects the performance of TRPDS(p) relative to TRPDS. In fact, TRPDS(p) takes 43 percent fewer concurrent function evaluations than TRPDS when $p = n + 1$. The difference is not nearly as substantial when $p = 2 \times n$. It is clear from the results that TRPDS requires fewer concurrent function evaluations when $p = 2 \times n$. This is not surprising for the following reason. Recall that the search scheme size is fixed at $2 \times n$. So when $p = n + 1$, TRPDS requires two concurrent function evaluations per PDS iteration, whereas when $p = 2 \times n$, only one concurrent function evaluation is required per PDS iteration. On the other hand, TRPDS(p) was designed so that only one concurrent function evaluation per PDS iteration is required, regardless of the number of processors. This accounts for the different relative performances of the two algorithms on different numbers of processors, and it brings to light the interesting observation described in the following paragraph.

Figure 5 shows a comparison of TRPDS(p) for two different values of p . In particular, we look at problems with $n = 8$ and compare performance for $p = n + 1$ and $p = 2 \times n$. Notice that setting $p = 2 \times n$ yields little, if any, improvement over $p = n + 1$. This is because the trial step is usually chosen from the $n + 1$ best points as determined by the quadratic model. Thus, the extra processors are being wasted on function evaluations that will not be used. While we do not expect this to be true for all models or problems, it does indicate that we must be careful when determining how to use the processors available to us. For example, in this case, it may be more beneficial to use extra processors to speculatively evaluate a finite difference gradient at the most promising trial points. A careful study of this and related issues is deferred to future work.

Finally, we compare TRPDS(p) to the BFGS trust-region algorithm with

speculative gradients. We show results for tests with the number of processors equal to $n + 1$ and $2 \times n$.

In Figure 6 we see that the BFGS trust-region algorithm with speculative gradients requires fewer concurrent function evaluations than TRPDS(p). As stated before, the case with the number of processors equal to $n + 1$ is the optimal number of processors for speculative gradients, so we expect speculative gradients to be doing well here. However, TRPDS(p) demonstrates that it can be competitive with speculative gradients.

Figure 7 compares TRPDS(p) with the speculative gradient BFGS trust-region algorithm when the number of processors is $2 \times n$. Although the

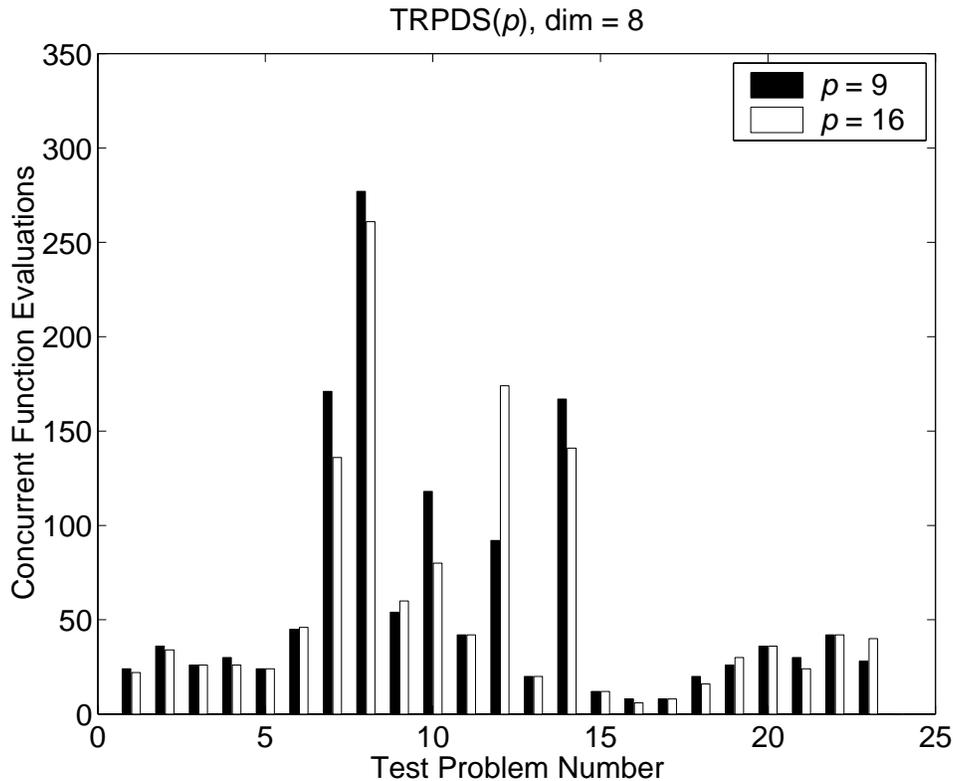


Figure 5: Comparison of TRPDS(p) with 9 and 16 processors. The dimension of the problem is 8. The extra parallelism does not improve the number of concurrent function evaluations for the TRPDS(p) algorithm. Perhaps the extra parallelism can be put to better use.

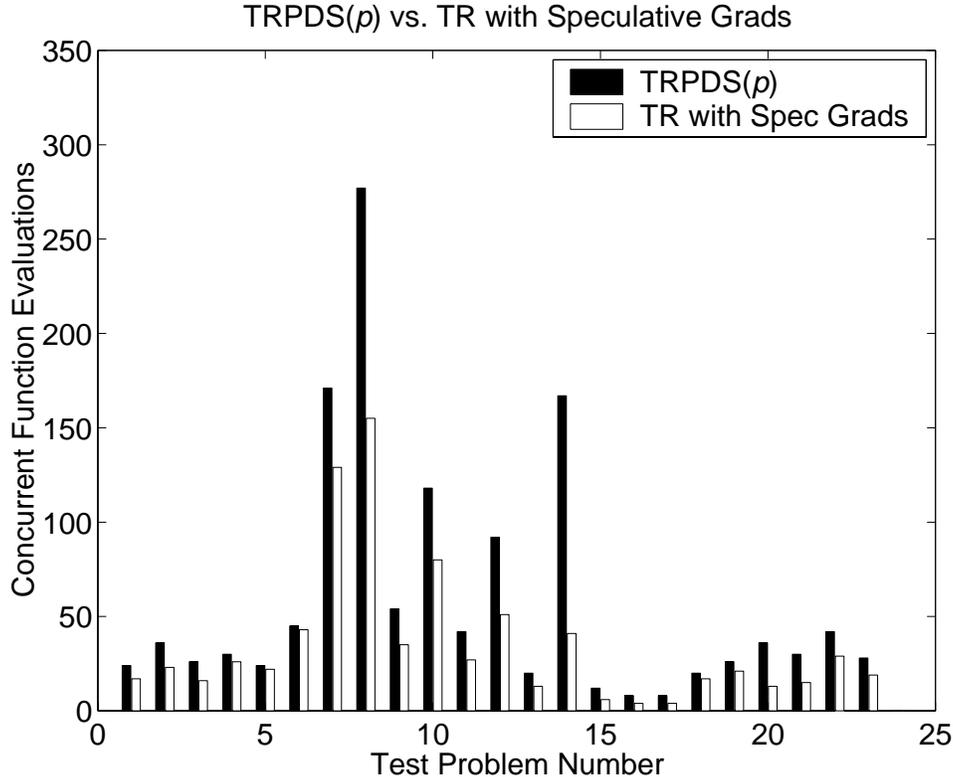


Figure 6: Comparison of TRPDS(p) and BFGS trust-region algorithm with speculative gradients. The dimension of the problem is 8, and the number of processors is 9. The number of concurrent function evaluations required by the BFGS trust-region algorithm with speculative gradients is less than the number required by the TRPDS(p) algorithm on every problem.

speculative gradient method still usually takes fewer concurrent function evaluations, TRPDS(p) is becoming more competitive. Note that the additional processors (more than $n + 1$) are not at all helpful to the speculative gradient approach. If we continued to add more and more processors, we could take advantage of them by also speculatively computing components of a finite difference Hessian. Beyond that, however, the speculative gradient method could make no further use of additional processors. On the other hand, TRPDS(p) could use these processors, not only for speculative computation, but also for other enhancements, some of which will be discussed in section 5. Thus, we expect the flexibility of TRPDS(p) to allow it not only

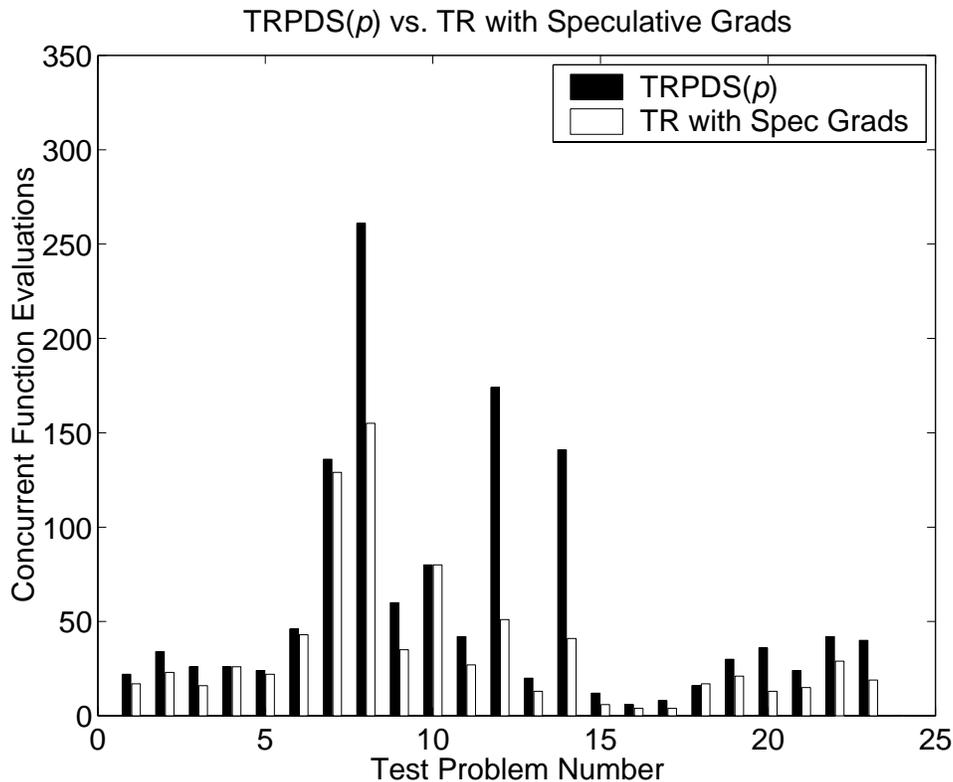


Figure 7: Comparison of TRPDS(p) and BFGS trust-region algorithm with speculative gradients. The dimension of the problem is 8, and the number of processors is 16. In this case, the BFGS trust-region algorithm with speculative gradients beats TRPDS(p) on most problems. The three exceptions are problems 4, 10, and 18.

to catch up with the BFGS trust-region with speculative gradients, but to surpass it.

4.2 Furnace design test problem - TWAFER

As a second example, we chose an optimal control problem for a vertical, multi-wafer furnace. Vertical furnaces can process up to 200 silicon wafers in a single batch and have been used for thin film deposition, oxidation, and other thermal process steps. The evolution of vertical furnaces has been driven by the need for process uniformity (that is, wafer-to-wafer and within-

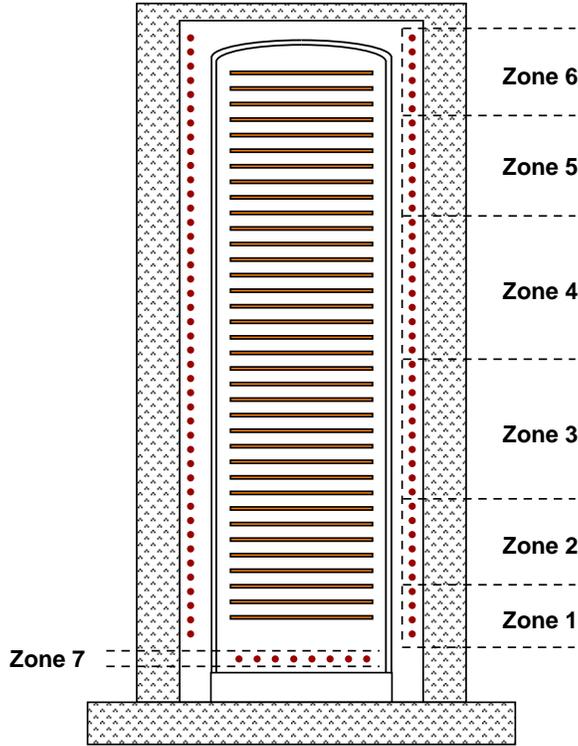


Figure 8: *Vertical Batch Furnace with Seven Control Zones*

wafer uniformity) and high wafer throughput. A recent variation of the multi-wafer reactor design is the small-batch, fast-ramp (SBFR) furnace. The SBFR is designed to heat-up and cool-down quickly, thus reducing cycle time and thermal budget. The SBFR consists of a stack of 50 eight-inch (diameter) silicon wafers enclosed in a vacuum-bearing quartz jar. The stack is radiatively heated by resistive coil heaters contained in an insulated canister. The heating coils can be individually controlled or ganged together in zones to vary the emitted power along the length of the reactor; a seven-zone configuration is shown in Figure 8. There are six control zones (each containing several heating coils) along the length of the furnace and one heater zone in the base. The zones near the ends of the furnace are usually run hotter than the middle zones to make up for heat loss.

The thermal optimal control problem can be described as follows. Given a set number of heating coils in a fixed zone configuration, find the optimal power settings such that the temperature uniformity about a fixed set-point

is maximized. The objective function, f , is defined by a least-squares fit of the N discrete wafer temperatures, T_i , to a prescribed temperature, T^* ,

$$F(\mathbf{p}) = \sum_{i=1}^N (T_i - T^*)^2 \quad (4)$$

where the \mathbf{p} are the unknown power parameters.

The engineering heat transfer model used in this example was developed by Houf [6] specifically for the analysis of vertical furnaces (the actual simulation code used in our experiments is called TWAFER). The heat transfer formulation is simplified by using mass lumping and one-dimensional approximations. The nonlinear transport equations are solved using the TWOPNT solver [5], which uses a Newton method with a time evolution feature. There

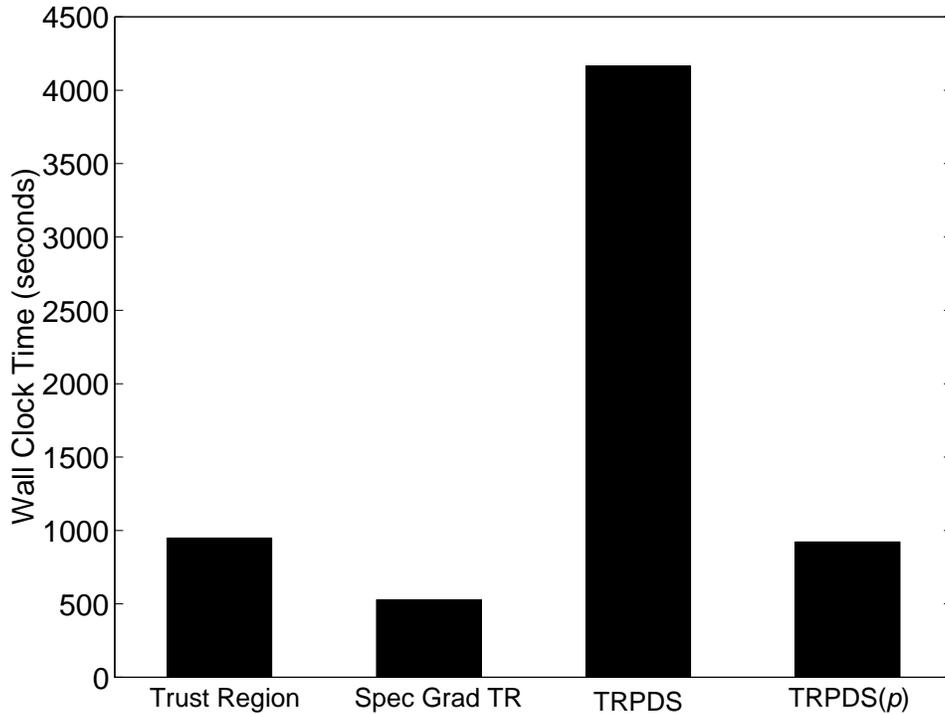


Figure 9: Comparison of standard trust-region method, trust-region with speculative gradients, TRPDS, and TRPDS(p) on 7-Zone TWAFER with 8 Processors. The BFGS trust-region algorithm with speculative gradients is the fastest algorithm.

are many different parameter combinations that have been considered in previous studies of the TWAFER code [8]. For this particular example we used only one configuration, namely a problem with 7 heater zones: one bottom heater and six equally-sized side heaters. Each simulation used a model that contained 100 wafers with ten discretization points per wafer. Our initial guess for the powers was: $\mathbf{p} = [100, 200, 300, 2700, 100, 400, 2000]$.

Figures 9 and 10 compare the total wall clock time required for solution by the standard trust-region, BFGS trust-region with speculative gradients, TRPDS, and TRPDS(p) algorithms. Comparisons are shown with $n + 1$ and $2 \times n$ processors.

As in the previous test problems, going from a standard BFGS trust-region algorithm to one using speculative gradients gives us approximately

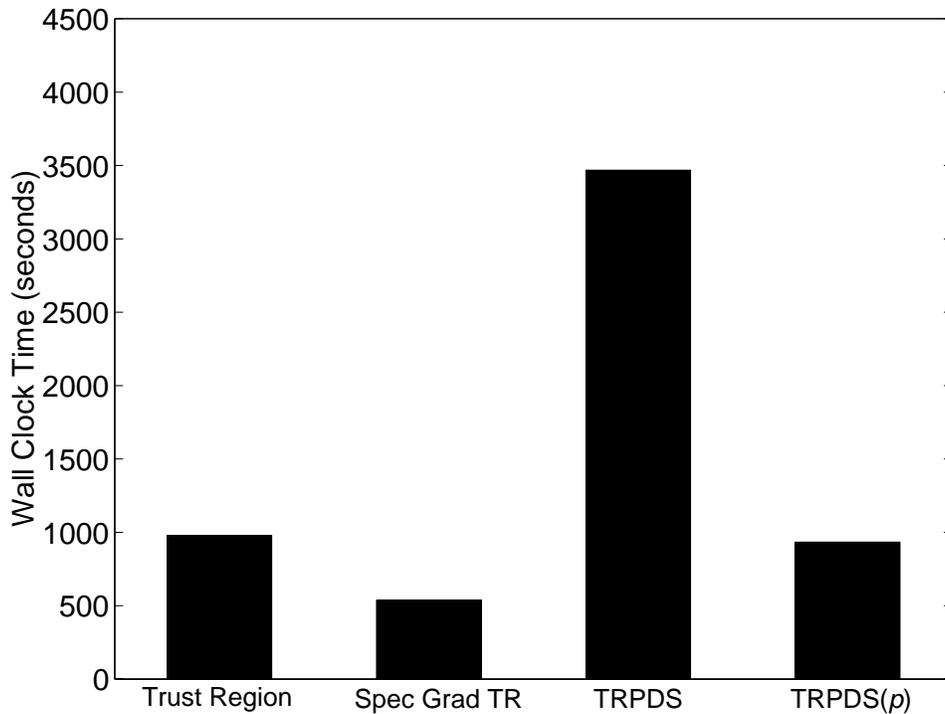


Figure 10: Comparison of standard trust-region method, trust-region with speculative gradients, TRPDS, and TRPDS(p) on 7-Zone TWAFER with 14 Processors. TRPDS(p) takes the same amount of time to solution as with 8 processors. Perhaps the extra processors can be put to better use.

44 percent improvement in wall clock time in this case, and TRPDS(p) gives nearly 80 percent improvement over TRPDS. The BFGS trust-region algorithm with speculative gradients is significantly faster in wall clock time than the standard TRPDS algorithm and somewhat faster than TRPDS(p). Note however, that the number of processors used in this example is $n+1$, the ideal case for speculative gradients. Furthermore, TRPDS is doing two concurrent function evaluations per iteration in this case.

In Figure 10, we have $2 \times n$ processors. Notice that while the speculative gradient algorithm and TRPDS(p) algorithm perform exactly as they did on the smaller number of processors in Figure 9, TRPDS is starting to catch up. This is not surprising for the reasons already discussed with the standard test problems. We emphasize again that because of its flexibility, we expect that TRPDS(p) can be modified to make better use of additional processors, yielding an algorithm faster than the speculative gradient approach.

5 Conclusions

We have added an option to calculate speculatively the gradient in the standard BFGS trust-region algorithm. Furthermore, we have added a new phase to the solution of the PDS subproblem in the TRPDS algorithm that takes advantage of an approximation model. Both of these options take advantage of parallel processing. These two options were tested on a standard set of test problems and on an engineering application. On these test problems, the BFGS trust-region method with speculative gradients tended to do better than the non-speculative variant, requiring up to 50 percent fewer concurrent function evaluations and shorter wall clock time. Similarly, TRPDS(p) generally performed better than the standard TRPDS algorithm, requiring an average of 36 percent fewer concurrent function evaluations and shorter wall clock time. On these problems, the BFGS trust-region method with speculative gradients tended to perform better than TRPDS(p) using the same performance criteria. However, TRPDS(p) is close enough to show promise, and it offers a great deal of flexibility for future performance-enhancing modifications. We discuss our future plans below.

In order to improve the TRPDS(p) algorithm, we would first like to determine which parameters have the greatest impact on the performance of the algorithm. We expect to use techniques from the design and analysis of computer experiments (DACE) literature in order to conduct this study.

Another factor to consider is that, like the traditional TRPDS algorithm, we expect TRPDS(p) to be robust in the presence of noise. The results of Hough and Meza [7] show that TRPDS outperforms a standard trust-region method in the presence of noisy function values and gradient approximations. We would like to compare TRPDS(p) and the BFGS trust-region method with speculative gradients on a problem with noisy functions.

In order to take advantage of the great flexibility of TRPDS(p), we would like to incorporate different types of approximation models, not only in the PDS phase of the algorithm, but also within the trust-region framework. Although the quadratic model has served us well, there may be approximation models more suitable for the applications of interest. Alexandrov, Dennis, Lewis, and Torczon [1] cite several examples that we would like to try. With multiple models to choose from, we would like to come up with a model management framework, as suggested in [1], where the approximation model could change at every iteration. We would also like to consider alternatives to the PDS subproblem with other computationally inexpensive approaches.

Finally, we intend to extend our work with speculative computations. In particular, we would like to incorporate a speculative Hessian evaluation in the standard trust-region method. We would also like to incorporate speculative gradient evaluations in TRPDS(p).

Acknowledgments: The authors wish to thank John Dennis, Juan Meza, and Virginia Torczon for many helpful discussions and suggestions.

References

- [1] N. ALEXANDROV, J. E. DENNIS, JR., R. M. LEWIS, AND V. TORCZON, *A trust region framework for managing the use of approximation models in optimization*, Structural Optimization, 15 (1998), pp. 16–12.
- [2] R. H. BYRD, R. B. SCHNABEL, AND G. A. SHULTZ, *Parallel quasi-newton methods for unconstrained optimization*, Mathematical Programming, 42 (1988), pp. 273–306.
- [3] A. R. CONN, N. I. M. GOULD, AND P. L. TOINT, *Testing a class of methods for solving minimization problems with simple bounds on the variables*, Tech. Rep. Research Report CS-86-45, University of Waterloo, Waterloo, CA, 1986.

- [4] P. E. GILL, W. MURRAY, AND M. H. WRIGHT, *Practical Optimization*, Academic Press, San Diego, CA, 1981.
- [5] J. F. GRGAR, *Twopnt program for boundary value problems, version 3.10*, Tech. Rep. SAND91-8230, Sandia National Laboratory, Livermore, CA, April 1992.
- [6] W. G. HOUF, J. F. GRGAR, AND W. G. BREILAND, *A model for low pressure chemical vapor deposition in a hot-wall tubular reactor*, Materials Science Engineering, B, Solid State Materials for Advanced Technology, 17 (1993), pp. 163–171.
- [7] P. D. HOUGH AND J. C. MEZA, *A class of trust-region methods for parallel optimization*, Tech. Rep. SAND99-8245, Sandia National Laboratories, Livermore, CA, 1999.
- [8] C. D. MOEN, P. A. SPENCE, AND J. C. MEZA, *Automatic differentiation for gradient-based optimization of radiatively heated microelectronics manufacturing equipment*, in Proceedings of 6th AIAA/NASA/ISSMO Symposium on Multidisciplinary Analysis and Optimization, Bellevue, WA, Sept 4–6 1996.
- [9] J. J. MORÉ, B. S. GARROW, AND K. E. HILLSTROM, *Testing unconstrained optimization software*, ACM Transactions on Mathematical Software, 7 (1981), pp. 17–41.
- [10] R. B. SCHNABEL, *A view of the limitations, opportunities, and challenges in parallel nonlinear optimization*, Parallel Computing, 21 (1995), pp. 875–905.